Adaptive Weight-Based Energy-Efficient Scheduling Algorithm for heterogeneous computing systems

Cheng Xu, Pan Shu, Tao Li, Yan Liu
College of Information Science and Engineering
Hunan University
Changsha, China
supermanpanshu@gmail.com

Abstract—**Energy-saving scheduling algorithm for parallel applications on heterogeneous computing systems has become an important research subject. Considering that the existing energy-efficient scheduling algorithms have strong locality and cannot flexibly adapt to the application performance (makespan /schedule length) requirements, the authors designed a weighted objective function, based on which an adaptive weight-based energy-efficient scheduling algorithm has been proposed with dynamic voltage scaling (DVS). It can effectively balance performance and power consumption by controlling the weight. The algorithm consists of two parts: (1) automatically calculate the optimum weight, thus consume less energy while guaranteeing makespan requirement; (2) use objective function to get the approximately optimal task allocation on the DVS-enabled processors through the idea of list scheduling algorithm. Compared to the other three existing task scheduling algorithms, the experimental results show that the new algorithm can much effectively balance schedule lengths and energy consumption.**

Keywords-heterogeneous computing system; dynamic voltage scaling (DVS); energy-efficient scheduling; green computing

I. INTRODUCTION

Over the years, heterogeneous computing systems have been widely used for compute-intensive and data-intensive applications. Notably, the energy consumption of heterogeneous computing systems is huge. According to the current study [1], the power consumption by computing centers accounted for about 0.5% of the world's total electricity consumption. The study also indicates that electricity consumption is expected to double by 2020. Clearly, there are environment issues with the generation of electricity [2]. Therefore, green energy has become one of the important factors that must be considered in high-performance computing.

Due to the importance of energy consumption, various techniques have been investigated and developed [3]. DVS (dynamic voltage scaling) among these has been proven to be a very promising technique with its demonstrated capability for energy savings (e.g., [4], [5], and [6]). DVS enables processors to dynamically adjust voltage supply levels aiming to reduce power consumption; however, this reduction is achieved at the expense of sacrificing clock frequencies.

Traditionally, the primary performance goal of heterogeneous computer systems has focused on reducing the execution time of applications. List scheduling algorithm [7] is a well-known algorithm for this performance goal, and has been studied separately with DVS. For reducing makespan the conventional list scheduling ignores that high-frequency and high voltage lead to high energy consumption. Although some algorithms by proposing a novel target function combined list scheduling algorithm and DVS to reduce the energy consumption [8], but these objective functions of scheduling algorithms does not consider the impact of a task completion time on the total energy consumption, so they are localized strongly. In addition, they also cannot be automatically adjusted according to the application performance requirement.

By analyzing the relationship of task completion time and energy consumption, we found that the energy consumption can be saved by executing task in lower voltage which will lead to extend the task completion time, however, because it can increase idle time, if left unchecked, the total energy consumption will increase instead. Thus this paper proposes a weighted objective function and comes up with an adaptive weight-Based energy-efficient scheduling algorithm (AWES). It combines list scheduling algorithm and DVS. AWES is essentially different from the existing scheduling algorithms. Firstly, with the relationship between task completion time and total energy consumption a novel target function with weight is proposed; secondly the optimal weight can be automatically calculated based on the performance requirements; finally, we can reduce energy consumption on the premise that makespan requirements can be met.

II. RELATED WORK

Due to the NP-hard nature of the task scheduling problem [9], heuristics are the most popular scheduling model adopted by many researchers. And for low complexity and high effect, the HEFT which is a well-known list-scheduling heuristic is widely used [7]. However, it ignores the energy problem. To solve this problem, LEE in [8] presented ECS and ECS+idle scheduling algorithm. The performance of these algorithms is very compelling in terms of both application completion time and energy consumption. But there are still lots of space for improvement.

Many algorithms have been developed for energy conservation, DVS is an important part of their. In [4] several different scheduling algorithms using the concept of slack sharing among DVS-enabled processors were proposed. Ma Yan et al. in [5] developed an algorithm based on integer linear programming (ILP) that chooses frequency and voltage level for executing parallel tasks. In [6] an adaptive threshold-based task duplication strategy was presented, it can meet the makespan requirement while reducing energy consumption. However, these previous studies on scheduling that take into consideration energy consumption are conducted on homogeneous computing systems.

III. MODELS

A. System Model

In this paper, the target system consists of a set P of m heterogeneous processors that are fully interconnected with high-speed network. Each processor is DVS enabled; in other words, processing unit P_j has a number of discrete voltage levels, which are given by $v_{j,k}$, k=1, 2, $N(j)$, where $N(j)$ denotes the total number of discrete voltage levels of P_j. The processor frequency of P_j at voltage level (VL) $v_{j,k}$ is given by $f_{j,k}$. Since clock frequency transition overheads take a negligible amount of time (e.g., 10us-150us [10]), these overheads are not considered in our study. Interprocessor communications are assumed to perform with the same speed on all links without contentions.

B. Application Model

Parallel application with a set of precedence-constrained tasks can be represented in form of a directed acyclic graph (DAG). A DAG, G= (N, E), consists of a set N of n nodes and a set E of e edges. The nodes represent tasks partitioned from an application; the edges represent precedence constraints and intertask communication. A task with no predecessors is called an entry task, n_{entry}, whereas an exit task, n_{exit}, is one that does not have any successors. Among the predecessors of a task n_i, the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the task denoted as MIP (i). The longest path of a task graph is the critical path (CP).

The computation cost of the task n_i on a processor P_j is denoted as $w_{i,j}$. The weight on an edge, denoted as $c_{i,j}$ represents the communication cost between two tasks, n_i and n_j. However, a communication cost is only required when two tasks are assigned to different processors. In other words, the communication cost when tasks are assigned to the same processor can be ignored. The earliest start and earliest finish times of, a task n_i on a processor P_j are defined as:

$$EST(n_i, P_j) = \begin{cases} 0 & n_i = n_{entry} \\ EFT(\mathrm{MIP}(i), P_k) + c_{\mathrm{MIP}(i),i} & \text{otherwise} \end{cases} \quad (1)$$

$$EFT(n_i, P_j) = EST(n_i, P_j) + w_{i,j} \quad (2)$$

where P_k is the processor on which the MIP of task n_i is scheduled.

C. Energy Model

CMOS devices are the building blocks of most general purpose computing systems today. Power consumed in CMOS circuits can be divided into three components: dynamic, static and short-circuit power. The dynamic power dissipation is the most significant factor of the power consumption. Dynamic power can be approximated with the following formula:

$$P_d = ACV^2 f \quad (3)$$

where A is the average number of circuit switches per clock cycle, C is the load capacitance, V is the supply voltage and f is the clock frequency.

If $PN_{j,N(j)}$ denotes the highest power, $v_{j,N(j)}$ denotes the highest voltage, and $f_{j,N(j)}$ denotes the highest frequency on processor P_j, then we can use (3) to compute the power when P_j executes with $v_{j,k}$ as:

$$PN_{j,k} = PN_{j,N(j)} \times \frac{f_j v_{j,k}^2}{f_{j,N(j)} v_{j,N(j)}^2} \quad (4)$$

Since processors consume a certain amount of energy while idling, the total energy consumption of the execution for a precedence-constrained parallel application in this study is comprised of direct and indirect energy consumption. The direct energy consumption is defined as:

$$E_d = \sum_{i=1}^{N} PN_{m(i),N(m(i))} \times \frac{v_{m(i),x(i,m(i))}^2}{v_{m(i),N(m(i))}^2} \times W_{i,m(i)} \quad (5)$$

where $m(i)$ is the processor on which task n_i executed, and $v_{m(i),x(i,m(i))}$ is the supply voltage of the processor $m(i)$. On the other hand, the indirect energy consumption is defined as:

$$E_i = \sum_{j=1}^{m} \sum_{idle_{j,k} \in IDLE_j} PN_{j,idle} \times t_{j,k} \quad (6)$$

where $IDLE_j$ is the set of idling slots on processor P_j, $PN_{j,idle}$ is the lowest power on P_j, and $t_{j,k}$ is the amount of idling time for $idle_{j,k}$. Then, the total energy consumption is defined as:

$$E_t = E_d + E_i \quad (7)$$

IV. ENERGY-CONSCIOUS SCHEDULING HEURISTICR

The AWES consists of two parts: (1) automatically calculate the optimum weight; (2) use objective function to get the approximately optimal task allocation. In addition, for further reducing the locality and sufficiently leveraging idle time, the MCER used in [12] is adopted.

A. Optimum Weight Calculation

The first phase in the AWES is to calculate the optimum weight.

Algorithm 1: Calculate_Optimal_ a

1: min_schedule_len←Task_Allocation(a =0)
2: max_schedule_len←Task_Allocation(a =4)
3: optimal_ a←4
4:According to the scope of makespan, set a makespan to meet the demand, denoted Assigned_Makespan
5: IF (Assigned_Makespan<min_schedule_len) THEN
6: optimal_ a←0
7: BREAK
8: ELSE
9: WHILE(optimal_ a >=0)
10: temp_sched_len←Task_Allocation(a= optimal_a)
11: IF (temp_sched_len>Assigned_Makespan) THEN
12: optimal_ a --
13: ELSE
14: BREAK
15: END IF
16: END WHILE
17:END IF
18:RETURN optimal_a

First, according to the allocation algorithm, we can conclude schedule length range (steps 1-2). Schedule length will increase as the weight, so the minimum makespan can be obtained when weight equals 0. And when weight is larger than a certain value (the experimental results obtained under 4), energy consumption will increase instead. Therefore, it is not necessary to use the weight larger than 4, the maximum makespan is obtained when weight equals 4. Steps 4 set a makespan requirement (Assigned_Makespan). If user setting is less than the minimum makespan, the optimal weight will be set as 0 (steps 5-7), otherwise, the weight from four starts diminishing, until the makespan is less than Assigned_Makespan or equals 0. In other words, once the system performance met, the loop terminates immediately, and set the optimal weight as the current weight.

B. Task Allocation

Scheduling algorithm must ensure the implementation of the predecessor task before the successor task execution. To satisfy this condition, we use b-level (e.g., [11]) to generate task allocation order. The b-level of a task is computed by adding the computation and communication costs along the longest path of the task from the exit task in the task graph. Note that, both computation and communication costs are averaged over all nodes and links.

Algorithm 2: Task_Allocation

1: Sort N in decreasing order by b-level value
2: for every n_i in N do
3: P_b←P_0 and V_b←$v_{0,0}$
4: for every P_j in P do
5: for every $v_{j,k}$ in V_j do
6: Compute R$(n_i, P_j, v_{j,k}, P_b, V_b)$
7: if R$(n_i, P_j, v_{j,k}, P_b, V_b)$ >0
8: n_i←P_j and V_b←$v_{j,k}$
9: end if
10: end for
11: end for
12: Assign n_i on P_b with V_b
13:end for
14:S←the current shedule
15:for every n_i in N do
16: Remove n_i in S
17: P_b←$P_{m(i)}$ and V_b←$v_{m(i),x(i,m(i))}$
18: for every P_j in P do
19: for every $v_{j,k}$ in V_j do
20: Recompute makespan
21: if no increase in makespan and $E_d(n_i, P_j, v_{j,k})$< $E_d(n_i, P_b, V_b)$
22: P_b←P_j and V_b←$v_{j,k}$
23: end if
24: end for
25: end for
26: Reassign n_i on P_b with V_b
27:end for

The R metric devised and incorporated into task allocation effectively deals with the trade-off of makespan and energy savings. Specifically, the R value of a scheduling alternative (task-processor-VL combination) is of a measure to identify the degree of energy efficiency relative to task execution time. For each scheduling combination in consideration, its R value is computed in addition to that of the best combination seen up to that point of decision making; that is, the latter is recomputed with the current combination being considered. The actual R value computation starts from the second combination due to the involvement of two combinations in each computation. A positive R value indicates the finding of a new best scheduling alternative. For a given task n_i, the R value of a scheduling combination of processor P_j and voltage $v_{j,k}$ with the best combination of P_b and V_b is defined as:

$$R(n_i, P_j, v_{j,k}, P_b, V_b) = \frac{E_d(n_i, P_b, V_b) - E_d(n_i, P_j, v_{j,k})}{P_{all}} \times a + EFT(n_i, P_b, V_b) - EFT(n_i, P_j, v_{j,k}) \quad (8)$$

where $E_d(n_i, P_j, v_{j,k})$ and $E_d(n_i, P_b, V_b)$ are the energy consumption of n_i on P_j with $v_{j,k}$ and that of n_i on P_b with V_b, the earliest finish time of the two task-processor

allocations are denoted as $EFT(n_i, P_j, v_{j,k})$ and $EFT(n_i, P_b, V_b)$, P_{all} is the total power of the entire system in the idle state, and a is the weight. For a given ready task, its R value with each pair of processor and voltage level is computed using the current best combination of processor and VL (P_b and V_b) for that task, and then the best combination—from which the maximum R value is obtained—is selected (steps 2-13).

Since each scheduling decision that AWES makes still tends to be confined to a local optimum, MCER is incorporated with the energy reduction phase without sacrificing time complexity (steps 15-27). It is an effective technique in lowering energy consumption, although the technique may not help schedules escape from local optima. For each task, MCER considers all of the other combinations of task, host and VL to check whether any of these combinations reduces the energy consumption of the task without increasing the current makespan.

V. Experiment & Analysis

This section presents the influence to the weight of the scheduling results, and compares AWES to existing three approaches which are HEFT, ECS and ECS+idle. HEFT is a well-known list-scheduling heuristic. ECS and ECS+idle are only two energy conscious scheduling algorithms for this situation.

The performance of AWES was thoroughly evaluated with a large set of random task graphs obtained by TGFF. A large number of variations were made on these task graphs for more comprehensive experiments, In addition to task graphs, various different characteristics of processors were applied to simulations. The random task graph set consisted of different graph sizes, CCRs (communication to computation ratio) and number of processors (2, 4, 6, and 8). 2000 graphs were randomly generated.

In this study, for a given task graph, we normalize both its makespan and energy consumption to lower bounds, the "schedule length ratio" (SLR) and "energy consumption ratio" (ECR) were used as the primary performance metrics. Formally, the SLR and ECR values of the makespan M and energy consumption E_t of a schedule generated for a task graph G by a scheduling algorithm are defined as:

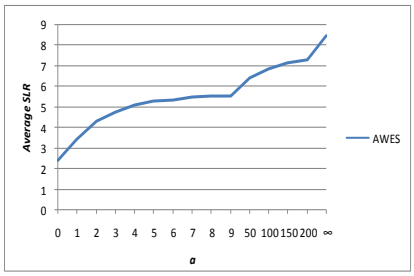
$$SLR = \frac{M}{\sum_{n_i \in CP} \min_{p_j \in P}\{w_{i,j}\}} \tag{9}$$

$$ECR = \frac{E_t}{\sum_{n_i \in CP} \min_{p_j \in P}\{w_{i,j} \times PN_{j,highest}\}} \tag{10}$$

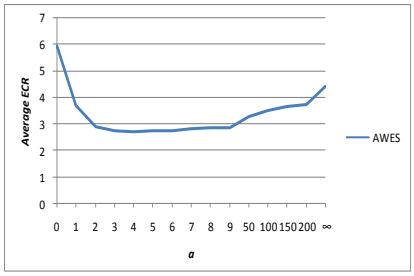
where CP is a set of CP tasks of G.

A. Impact of Weight

Now we investigate the impact on the scheduling as the weight a increases. As shown in fig.1a, the SLR increases with increasing a, and tends to a certain limit. The ECR decreases initially, then it begins to increase slowly and tends to a limit when a further increases (see fig.1b). This is because, the smaller energy consumption of the processor-VL combination has more chances to be selected by the increase of a. It causes that the voltage becomes lower, the task completion time increases, which eventually leads to increase makespan and decline the energy consumption at the beginning. As for the rebound of ECR, the growth of task completion time cause more idle time of processor, which can lead to the increasing amount of the indirect energy consumption exceeding the declined amount of the direct energy consumption.



(a)



(b)

Figure 1. Average SLR and ECR for different weight

B. Comparison Results

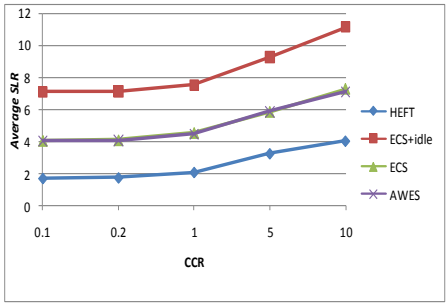
To prove the superiority of the objective function of this article, we set the weight as 4. The overall comparative results from our evaluation study are summarized in Table 1. Table 1 clearly signifies the superior performance of our algorithms over HEFT, ECS and ECS+idle. Specifically, schedules generated by AWES consumed on average 56 percent, 38. 6 percent and 10 percent less energy than HEFT, ECS and ECS+idle, respectively. In addition, our proposed algorithm mostly outperformed those three algorithms with various different CCRs as shown in Figs. 2.

TABLE I. Comparative Results

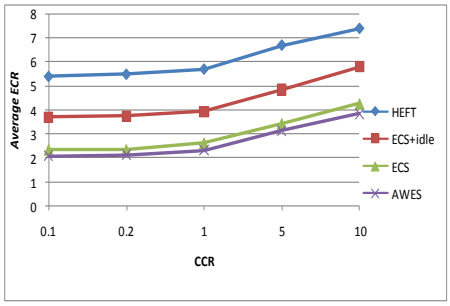
Algorithm	SLR	ECR
HEFT	2.578	6.151
ECS+idle	8.463	4.409
ECS	5.182	3.004
AWES	5.107	2.705

In many previous studies (e.g., [11]), HEFT has been proven to perform very competitively with a low time complexity. But it does not take energy consumption into consideration, so its energy consumption is much larger than the other three algorithms. And AWES can reach the same makespan of HEFT when *a* is 0, even when *a* takes 4, the makespan of AWES is only more than that of HEFT with 2.529, while less than ECS and ECS+idle with 0.075 and 3.356, respectively, so the SLR of AWES is compelling.

ECS and ECS+idle consider energy consumption in the objective function, so they show a good energy saving effect compared to HEFT. The reason why ECS+idle has poorer performance than ECS, is that the objective function value in ECS+idle is too dependent on energy. However, the objective functions of ECS and ECS+idle are too localized, so there is still lots of space for improvement. From table 1, we can find the superior performance of AWES over ECS and ECS+idle, in addition, our algorithm can get less makespan by requirement.



(a)



(b)

Figure 2. Average SLR and ECR for random DAGs

The source of the main performance gain of our algorithm is the use of the R objective function. It can effectively balance makespan and energy consumption. In our experiments, further 3 percent improvements (on average) in energy consumption—for schedules after the main scheduling phase of AWES—were made by the MCER technique.

VI. CONCLUSIONS

This paper has presented an adaptive weight-based energy-efficient scheduling algorithm for heterogeneous computing systems, suitable for DVS-enabled heterogeneous computing systems designed to reduce energy consumption on the premise that the makespan requirement is met. First, according to the makespan requirement, the algorithm dynamically adjusts and gets the optimal weight. Then with the objective function based on the optimal weight, it could work out a near-optimal solution by balancing the makespan and energy saving. Thereby, the scheduling results meet the makespan requirement while reducing energy consumption.

The experiments results show that, compared with other existing similar algorithm, ATWS can not only maintain a good schedule length, but also save a lot of energy.

ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China (Grant No.61272062 and No.61300037)

REFERENCES

[1] LIN Chuang, TIAN Yuan, YAO Min, "Green network and green evaluation: Mechanism, modeling and evaluation," Chinese Journal of Computers, vol. 34, no. 4, pp.593−612, 2011.

[2] GUO Bing, SHEN Yan, SHAO Zi-Li, "The redefinition and some discussion of green computing," Chinese Journal of Computers, vol. 32, no. 12, pp. 2311-2319, 2009.

[3] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," ACM Computing Survey, vol. 37, no. 3, pp. 195-237, 2005.

[4] D. Zhu, R. Melhem, and B.R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," IEEE Trans. Parallel and Distributed Systems, vol. 14, no. 7, pp. 686-700, July 2003.

[5] Yan Ma, Bin Gong, LiDa Zou, "Energy-optimization scheduling of task dependent graph on DVS-enabled cluster system," Proceedings of the 5th Annual ChinaGrid Conference(ChinaGrid), pp. 183−190,2010.

[6] LIU Wei, Yin Hang, DUAN Yu-Guang, Du Wei, WANG Wei, ZENG Guo-Sun, "Adaptive threshold-based energy-efficient scheduling algorithm for parallel tasks on homogeneous DVS-enabled clusters," Chinese Journal of Computers, vol. 36, no. 2, pp. 393-407, 2013.

[7] H. Topcuouglu, S. Hariri, Wu.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel and Distributed Systems, vol. 13,no. 1, pp. 260-274, Mar. 2002.

[8] Young Choon Lee, Albert Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 8, pp.1374−1381, 2011.

[9] M. R. Garey and D. S. Johnson, Computers and intractability: a guide to the theory of NP-Completeness. New York: W. H. Freeman, 1979.

[10] Intel, Intel Pentium M Processor Datasheet, 2004.

[11] S.C. Kim, S. Lee, J. Hahm, "Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster Systems," IEEE Trans. Parallel and Distributed Systems, vol. 18, no. 11, pp. 1489-1502, Nov. 2007.